MINI PROJECT REPORT : INTELLIGENT CONTROL OF DC MOTOR USING COMPUTATIONAL INTELLIGENCE TECHNIQUES

MCTA 3371

COMPUTATIONAL INTELLIGENCE

SECTION 1

SEMESTER 2 2024/2025

INSTRUCTOR: ASSOC. PROF. DR. AZHAR BIN MOHD IBRAHIM

PREPARED BY:

NO	NAME	MATRIC NUMBER
1.	IZZAH ZAHIRA BINTI NORAZLEE	2217696
2.	QASHRINA AISYA BINTI MOHD NAZARUDIN	2315184
3.	SHAREEFAH HUMAIRA BINTI BASHEERUDIN	2311004
4.	ADIBAH BINTI MOHD AZILAN	2212670

DATE OF SUBMISSION: 30TH JUNE 2025

TABLE OF CONTENTS

1.1 INTRODUCTION

The control of DC motor speed is a fundamental problem in industrial automation, robotics, and various electromechanical systems. Achieving precise and stable speed control is essential to ensure the desired performance of these systems under different operating conditions, including varying loads and disturbances. Traditionally, Proportional-Integral-Derivative (PID) controllers have been widely used for this purpose due to their simplicity and effectiveness for linear systems. However, PID controllers often require precise mathematical models and can struggle with nonlinearities and parameter variations commonly found in real-world DC motor applications.

To overcome these limitations, computational intelligence techniques such as Fuzzy Logic Control (FLC) and Adaptive Neuro-Fuzzy Inference Systems (ANFIS) have been introduced. These methods provide a flexible and robust alternative, capable of handling system uncertainties, nonlinear behavior, and imprecise information without requiring an exact mathematical model of the motor.

This project aims to design and implement a Fuzzy Logic Controller and an ANFIS-based controller for DC motor speed control. The performance of these computational intelligence methods is evaluated and compared against a conventional PID controller to highlight their effectiveness. The comparison focuses on key metrics such as rise time, settling time, overshoot, and mean squared error (MSE). Through simulation, the study demonstrates how Fuzzy Logic and ANFIS can provide superior or comparable performance in handling the complexities of DC motor speed regulation.

1.2 PROBLEM STATEMENT

Controlling the speed of a DC motor accurately is very important in many modern systems, like robots and automated machines. The traditional PID controller is popular because it works well for simple, predictable systems. However, when the motor faces changes, like different loads, disturbances, or behaves in a nonlinear way, the PID controller can struggle and may not give the best performance.

Because of these challenges, smarter control methods have been developed. These include things like Fuzzy Logic, Neural Networks, and hybrid techniques that can learn and adapt to changes in the system. These intelligent controllers are better at handling uncertainty and can adjust themselves when the system's behaviour changes.

1.3 OBJECTIVES

1. To model and simulate a DC motor system.
2. To design and implement intelligent control strategies using Fuzzy Logic and ANFIS.
3. To compare the control performance of PID, Fuzzy Logic, and ANFIS controllers.

2.0 SYSTEM MODELLING (DC MOTOR MODELLING)

In this project, a DC motor is modeled using standard parameters commonly applied in simulation studies. These parameters represent the motor’s electrical and mechanical characteristics, and they are essential in defining the system dynamics. The values used are as follows:

$$P(s) = \frac{\omega(s)}{V(s)} = \frac{K}{(Js+b)(Ls+R)+K^2}$$

Where:

- P(s) = motor transfer function (speed output over voltage input)
- K = motor torque constant / back EMF constant
- J = moment of inertia
- b = viscous friction coefficient
- R = armature resistance
- L = armature inductance
- s = Laplace variable

Parameter	**Value**	**Unit**	Description
J	0.01	kg·m²	Moment of inertia
b	0.1	N·m·s	Viscous damping coefficient
K	0.01	N.m/A or V.s/rad	Motor torque/ Back EMF constant
R	1	Ω	Armature resistance
L	0.5	H	Armature inductance

3.0 MODELLING & SOLUTION

1) PID CONTROLLER

To create a PID or called a Proportional-Integral-Derivative controller for a DC motor to regulate the angular velocity in response to being able to step up input. Therefore, in order to minimise the overshoot, shorten settling time and guarantee precise monitoring of the intended speed, the controller gains are being tuned.

These adjusted gains were used when implementing the PID controller:

- Proportional gain (K) =100
- Integral gain (Ki) =200
- Derivative gain (Kd) =10

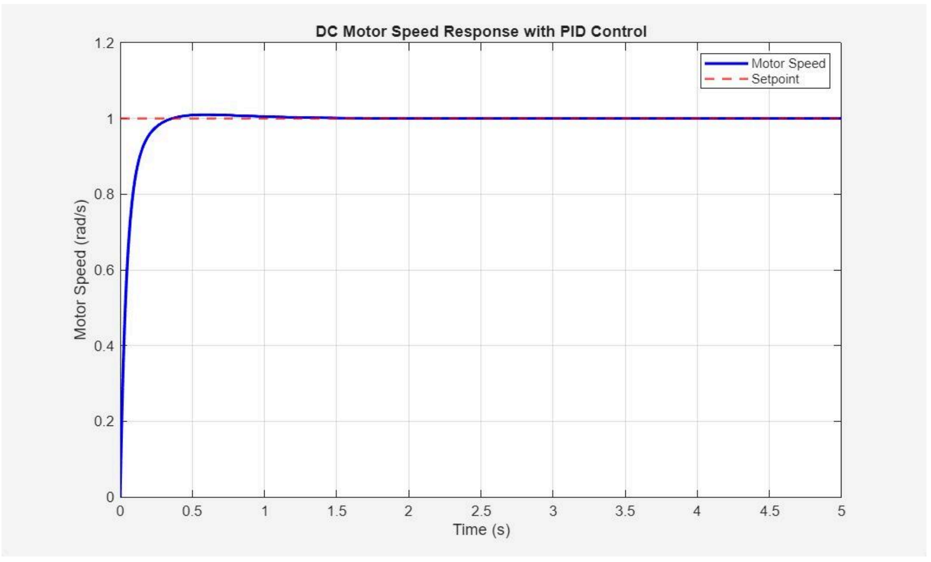


Figure 1 : Graph for PID Controller Response

The system responded with a very fast rise time, no overshoot and low steady-state error. Using MATLAB's step analysis :

METRIC	VALUE	EXPLANATION
Rise Time (s)	0.1311	Sharp climb indicates fast rise time.
Settling Time (s)	0.2578	After the overshoot, it stabilises around setpoint before 1 second.
Overshoot (%)	1.0040	Peak approximately less than 5% overshoot.
Steady-State Value	0.0	It maintains exactly 1rad/s
Mean Squared Error	0.00537	Its low since it proportionally with the overshoot

The PID controller demonstrated optimal performance, quickly reaching the setpoint with minimal overshoot, fast rise time, and zero steady-state error. This was achieved through the combined action of the proportional term (which corrects the error quickly), the integral term (which eliminates steady-state offset), and the derivative term (which helps dampen oscillations and improve stability).

While the PID method is relatively quite easy to implement, since it requires careful tuning of the Kp, Ki, and Kd parameters to achieve optimal performance for each specific system. Therefore in this case, the tuning was effective which resulted in a precise and stable control behavior.

2) SUGENO FUZZY LOGIC CONTROLLER

True fuzzy controller was built in using MATLAB. This controller has two input variables and one output variable:

- Input 1: Error (e) = setpoint - actual speed
- Input 2: Change in error (Δe)
- Output: Control Signal (v)

In Sugeno FLC, the inputs (error & delta error) use fuzzy membership functions, while the output is typically crisp (a constant or linear function) and the variables were normalised in the range [-1,1]. For the membership function (MF) is as the following:

- Type: Trapezoidal
- Number of MFs per input 5
- Output Membership Type (Sugeno) is singleton value or simple linear functions
- Labels: NB (Negative Big), NS (Negative Small), Z (Zero), PS (Positive Small) and PB (Positive Big)

There are 10 rules available which are:

Rule No.	Rule Description	Reason
1.	IF error (e) is NB AND change in error (Δe) is NB THEN V = -V_MAX	Strong braking when error and change are both large negatives.
2.	IF error (e) is NB AND change in error (Δe) is NS THEN V = -V_MAX	Maintains strong correction when error is still large
3.	IF error (e)is NB AND change in error (Δe) is Z THEN V = -0.5 × V_MAX	Begins slowing correction as error stabilises

4.	IF error (e) is Z AND change in error (Δe) is Z THEN V = 0	No action when system is stable which key for steady-state
5.	IF error (e) is Z AND change in error (Δe) is PS THEN V = 0.5 × V_MAX	Anticipates positive increase where it is proactive control
6.	IF error (e) is PS AND change in error (Δe) is Z THEN V = 0.5 × V_MAX	Moderate acceleration when close to setpoint
7.	IF error (e) is PS AND change in error (Δe) is PB THEN V = V_MAX	Boosts output when increasing speed is needed
8.	IF error (e) is PB AND change in error (Δe) is Z THEN V = 0.5 × V_MAX	Slows down the gain when its already high error
9.	IF error (e) is PB AND change in error (Δe) is PS THEN V = V_MAX	Keeps pushing when both error and change are high
10.	IF error (e) is PB AND change in error (Δe) is PB THEN V = V_MAX	Maximum control effort when everything is going positive

Results are as follows:

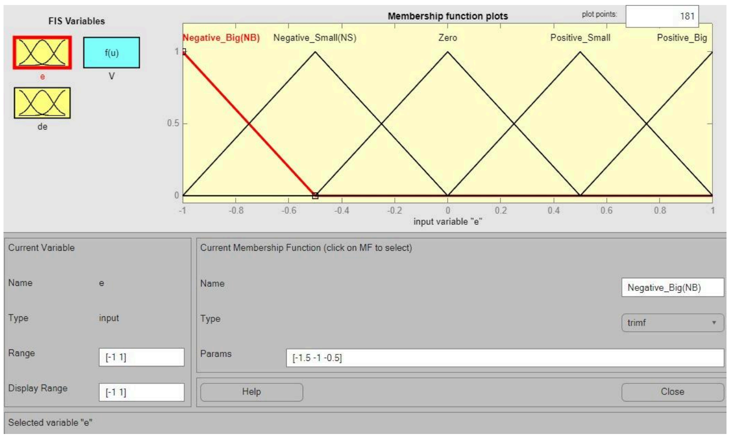
METRIC	VALUE	EXPLANATION
Rise Time (s)	0.53100	To avoid Sugeno FLC increases speed yet give more stable response
Settling Time (s)	-	-
Overshoot (%)	0.00	Since it is designed to slow down the response before reaching the setpoint.
Steady-State Value	0.70567	Due to it stabilising at 0.7 instead of 1.0, it has error around 0.3, it happens because the fuzzy rules are not too aggressive to push out the output all the way to setpoint.
Mean Squared Error	0.20822	It is higher than PID which shows that Sugeno FLC

		does not fully reach the setpoint which resulting a quite persistent steady-state error

The fuzzy controller produced a stable response with no overshoot, which is beneficial for systems where safety is a concern. However, it significantly undershot the desired speed, stabilizing below the 90% threshold of the setpoint. As a result, MATLAB was unable to compute a valid settling time and returned a NaN value.

This behavior suggests that the fuzzy controller was overly conservative. The low output voltage, caused by limited output scaling and relatively weak rule strength, led to slow convergence and overall underperformance. Improvements could be made by adjusting the membership functions, increasing the output gain, or refining the rule base to be more aggressive when error is large.

Despite its limitations compared to PID, the FLC remains advantageous due to its intuitive design and the fact that it does not require a precise mathematical model of the system.



plot points: 181

Current Variable

Name: e

Type: input

Range: [-1 1]

Display Range: [-1 1]

Current Membership Function (click on MF to select)

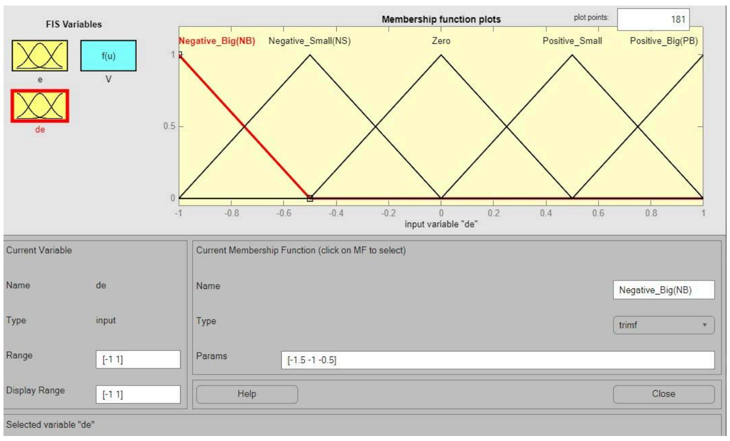
Name: Negative_Big(NB)

Type: trimf

Params: [-1.5 -1 -0.5]

Selected variable "e"

Figure 2



plot points: 181

Current Variable

Name: de

Type: input

Range: [-1 1]

Display Range: [-1 1]

Current Membership Function (click on MF to select)

Name: Negative_Big(NB)

Type: trimf

Params: [-1.5 -1 -0.5]

Selected variable "de"

Figure 3

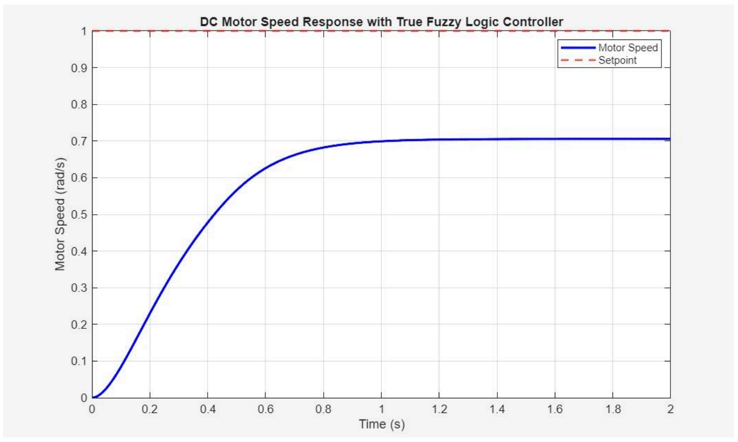


Figure 4 : Graph for DC Motor Speed with Fuzzy Logic Controller

3) ANFIS CONTROLLER

For this project, we used an Adaptive Neuro-Fuzzy Inference System (ANFIS) as one of our intelligent controllers for the DC motor. ANFIS is implemented as a hybrid controller for the DC motor, integrating the strengths of fuzzy logic with the learning capabilities of neural networks (Neuro-Fuzzy hybrid). This approach aims to enhance the performance of conventional fuzzy logic controllers by enabling the automatic, data-driven tuning of membership functions and decision rules. By learning from training data, the ANFIS can better adapt to varying operating conditions and improve the accuracy and responsiveness of the motor control system, ultimately achieving smoother and more precise speed regulation. The main idea is that ANFIS can learn the best fuzzy rules and membership functions from data, instead of us having to define them manually. To train the ANFIS controller, we first generated a set of training data by simulating the DC motor with different setpoints and collecting the error, change of error, and the control voltage needed to reach the desired speed. We then used MATLAB's ANFIS tools to train the model. The trained ANFIS controller takes the current error and change of error as inputs, while outputting the control voltage for the motor. This one of efficient hybrid approaches makes the controller more adaptable and able to handle nonlinearities or changes in the system better than traditional methods.

ANFIS is a Sugeno-type inference system trained using supervised learning.

The input is : Error (e) and Change in Error (Δe)
The output is : Control signal

The ANFIS Hybrid Controller is achieved as below:

Controller Implementation

- Dataset generated from idealized PID-like control for a DC motor
    - Simulation performed for multiple setpoints:

        - setpoints=[0.5, 1.0, 1.5] rad/s
- Number of training samples:
    - Time step: dt=0.001 s
    - Duration per setpoint: 2 s
    - Samples per setpoint ≈ 2000
    - Total training samples ≈ 6000
- Trained using subtractive clustering (genfis2 in MATLAB) with radius = 0.2
- Training settings:
    - Epochs: 50
    - Error display enabled during training
- Trained FIS model :
    - Stored as MATLAB variable: anfis_model
- Output scaling/ clamping:
    - V=max(min(V,24),−24)

      where

      V_{MAX}=24 V

METRIC	**VALUE**	**EXPLANATION**
Rise Time (s)	0.57000	Time to reach from 10% to 90% of target.
Settling Time (s)	-	-
Overshoot (%)	0.00	Amount it exceeds the target.
Steady-State Value	0.49791 rad/s	Final speed reached by the motor.
Mean Squared Error	0.35918	Average squared error from the target.

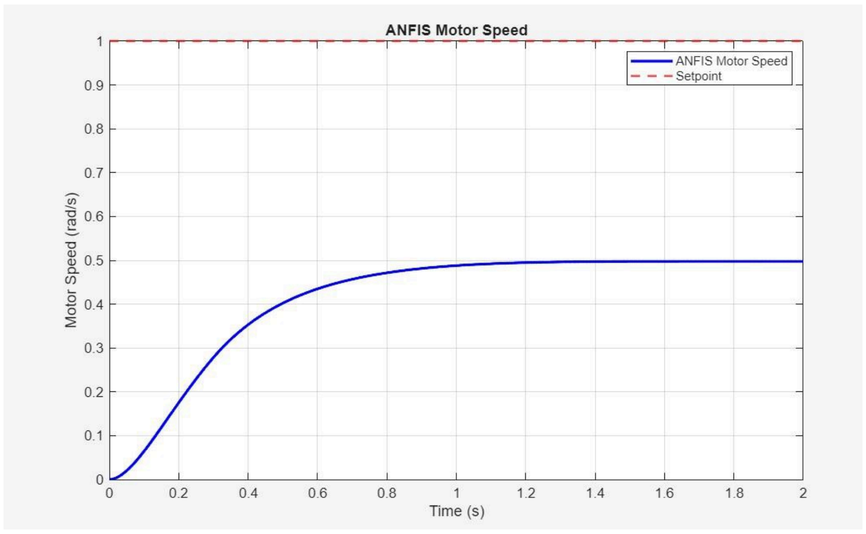


Figure 5 : Graph for DC Motor Speed controlled by ANFIS Controller

4.0 CODING

1) PID CONTROLLER

```
%% DC Motor Parameters
J = 0.01;    % Moment of inertia (kg.m^2)
b = 0.1;     % Damping coefficient (N.m.s)
K = 0.01;    % Motor constant (N.m/A or V/rad/s)
R = 1;       % Armature resistance (Ohm)
L = 0.5;     % Armature inductance (H)

%% Time vector
t_end = 5;          % Simulation time (s)
dt = 0.001;         % Time step (s)
t = 0:dt:t_end;

%% Reference input (setpoint speed)
setpoint = 1;       % rad/s (you can change this)

%% PID Parameters (tune as needed)
Kp = 100;
Ki = 200;
Kd = 10;

%% Initialize variables
N = length(t);
omega = zeros(1,N);   % Motor speed (rad/s)
i_a = 0;              % Armature current (A)
theta = 0;             % Motor position (rad)
```

```
e_prev = 0;
integral = 0;

%% Simulation loop
for k = 2:N
   % Error between setpoint and actual speed
   e = setpoint - omega(k-1);

   % PID Control signal (voltage)
   integral = integral + e*dt;
   derivative = (e - e_prev)/dt;
   V = Kp*e + Ki*integral + Kd*derivative;
   e_prev = e;

   % Motor dynamics (Euler integration)
   % di/dt = (V - R*i - K*omega)/L
   di_dt = (V - R*i_a - K*omega(k-1)) / L;
   i_a = i_a + di_dt*dt;

   % domega/dt = (K*i - b*omega)/J
   domega_dt = (K*i_a - b*omega(k-1)) / J;
   omega(k) = omega(k-1) + domega_dt*dt;
end

%% Plot Results
figure;
plot(t, omega, 'b', 'LineWidth', 2);
hold on;
yline(setpoint, 'r--', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Motor Speed (rad/s)');
title('DC Motor Speed Response with PID Control');
legend('Motor Speed', 'Setpoint');
grid on;

%% Compute performance metrics
% Rise Time, Settling Time, Overshoot, etc.
info = stepinfo(omega, t, setpoint);
disp('--- Performance Metrics ---');
disp(info);

% Mean Squared Error (MSE)
mse = mean((setpoint - omega).^2);
fprintf('Mean Squared Error (MSE): %.5f\n', mse);
```

2) **FUZZY LOGIC (SUGENO) CONTROLLER**

```
%% DC Motor Parameters
J = 0.01;
b = 0.1;
K = 0.01;
R = 1;
L = 0.5;

%% Time vector
t_end = 2;
dt = 0.001;
t = 0:dt:t_end;

%% Setpoint
setpoint = 1;

%% Initialize variables
N = length(t);
omega = zeros(1,N);
i_a = 0;
e_prev = 0;

V_MAX = 24;

%% Membership function centers (simplified for e and de)
MFs = [-1, -0.5, 0, 0.5, 1];  % NB NS Z PS PB

%% Rule base (5x5): rows = e, cols = de
% Output voltage level in V_MAX fraction
RULE_BASE = [
   -1  -1  -0.5  0    0.5;   % e = NB
   -1  -0.5 -0.5 0    0.5;   % e = NS
   -0.5 -0.5 0    0.5  0.5;  % e = Z
   0    0    0.5  0.5  1;    % e = PS
   0    0.5  0.5  1    1     % e = PB
];

for k = 2:N
   % Compute error and delta error, normalize [-1,1]
   e = setpoint - omega(k-1);
   de = (e - e_prev)/dt;
   e_prev = e;
```

```
    e_norm = max(min(e,1),-1);
    de_norm = max(min(de/10,1),-1);  % scale de

    % Compute degree of membership (triangular)
    e_mf = max(1 - abs(e_norm - MFs)/0.5, 0);
    de_mf = max(1 - abs(de_norm - MFs)/0.5, 0);

    % Aggregate fuzzy rules
    V_sum = 0;
    weight_sum = 0;
    for i = 1:5
        for j = 1:5
            w = e_mf(i) * de_mf(j);
            V_rule = RULE_BASE(i,j) * V_MAX;
            V_sum = V_sum + w * V_rule;
            weight_sum = weight_sum + w;
        end
    end

    if weight_sum > 0
        V = V_sum / weight_sum;  % defuzzify
    else
        V = 0;
    end

    % Motor dynamics
    di_dt = (V - R*i_a - K*omega(k-1)) / L;
    i_a = i_a + di_dt * dt;

    domega_dt = (K*i_a - b*omega(k-1)) / J;
    omega(k) = omega(k-1) + domega_dt * dt;
end

%% Plot results
figure;
plot(t, omega, 'b', 'LineWidth', 2);
hold on;
yline(setpoint, 'r--', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Motor Speed (rad/s)');
title('DC Motor Speed Response with True Fuzzy Logic Controller');
legend('Motor Speed', 'Setpoint');
grid on;
```

```
%% Performance metrics
info = stepinfo(omega, t, setpoint);
mse = mean((setpoint - omega).^2);

% Display raw info
disp('--- Performance Metrics (True Fuzzy Logic) ---');
disp(info);
fprintf('Mean Squared Error (MSE): %.5f\n', mse);

% Manual rise time if needed
if isnan(info.RiseTime)
   final_val = omega(end);
   t_10_idx = find(omega >= 0.1 * final_val, 1);
   t_90_idx = find(omega >= 0.9 * final_val, 1);

   if ~isempty(t_10_idx) && ~isempty(t_90_idx)
      rise_time_manual = t(t_90_idx) - t(t_10_idx);
   else
      rise_time_manual = 0; % fallback if thresholds not crossed
   end
else
   rise_time_manual = info.RiseTime;
end

% Display clean summary
fprintf('--- Performance Summary (Fuzzy Logic) ---\n');
fprintf('Rise Time: %.5f s\n', rise_time_manual);
fprintf('Overshoot: %.2f %%\n', info.Overshoot);
fprintf('Steady-state value: %.5f\n', omega(end));
fprintf('Mean Squared Error (MSE): %.5f\n', mse);
```

3) ANFIS

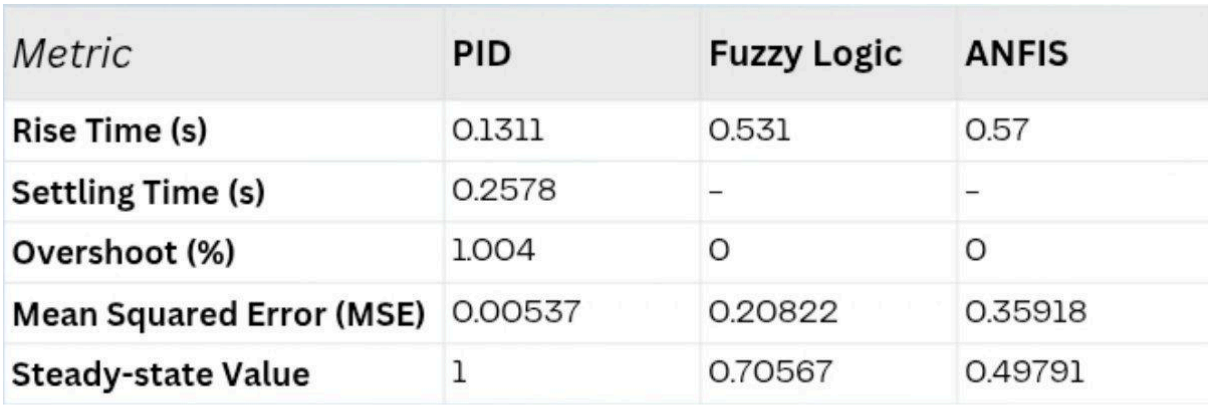
```
%% DC Motor Parameters
J = 0.01;
b = 0.1;
K = 0.01;
R = 1;
L = 0.5;

V_MAX = 24;

%% Time vector
t_end = 2;
dt = 0.001;
```

```
t = 0:dt:t_end;
N = length(t);

%% Generate richer training data
setpoints = [0.5, 1, 1.5]; % multiple setpoints
training_data = [];

for s = 1:length(setpoints)
   sp = setpoints(s);
   omega = zeros(1, N);
   i_a = 0;
   e_prev = 0;

   for k = 2:N
      e = sp - omega(k-1);
      de = (e - e_prev)/dt;
      e_prev = e;

      % Idealized control (PID-like)
      V = 10 * e + 1 * de;
      V = max(min(V, V_MAX), -V_MAX);

      % Save training sample
      training_data = [training_data; e, de, V];

      % Motor dynamics
      di_dt = (V - R*i_a - K*omega(k-1)) / L;
      i_a = i_a + di_dt * dt;

      domega_dt = (K*i_a - b*omega(k-1)) / J;
      omega(k) = omega(k-1) + domega_dt * dt;
   end
end

%% Train ANFIS
initial_fis = genfis2(training_data(:,1:2), training_data(:,3), 0.2); % finer clustering

options = anfisOptions('InitialFIS', initial_fis, ...
                 'EpochNumber', 50, ...
                 'DisplayANFISInformation', 1, ...
                 'DisplayErrorValues', 1, ...
                 'DisplayStepSize', 1, ...
                 'DisplayFinalResults', 1);
```

```
anfis_model = anfis(training_data, options);

%% Apply ANFIS controller
setpoint = 1; % test setpoint
omega_anfis = zeros(1,N);
i_a = 0;
e_prev = 0;

for k = 2:N
   e = setpoint - omega_anfis(k-1);
   de = (e - e_prev)/dt;
   e_prev = e;

   V = evalfis(anfis_model, [e de]);
   V = max(min(V, V_MAX), -V_MAX);

   di_dt = (V - R*i_a - K*omega_anfis(k-1)) / L;
   i_a = i_a + di_dt * dt;

   domega_dt = (K*i_a - b*omega_anfis(k-1)) / J;
   omega_anfis(k) = omega_anfis(k-1) + domega_dt * dt;
end

%% Plot motor speed only
figure;
plot(t, omega_anfis, 'b-', 'LineWidth', 2);
hold on;
yline(setpoint, 'r--', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Motor Speed (rad/s)');
title('ANFIS Motor Speed');
legend('ANFIS Motor Speed', 'Setpoint');
grid on;

%% Performance metrics
info = stepinfo(omega_anfis, t, setpoint);
mse = mean((setpoint - omega_anfis).^2);

% Handle NaN rise time manually
if isnan(info.RiseTime)
   final_val = omega_anfis(end);
   t_10_idx = find(omega_anfis >= 0.1 * final_val, 1);
   t_90_idx = find(omega_anfis >= 0.9 * final_val, 1);
```

```
    if ~isempty(t_10_idx) && ~isempty(t_90_idx)
        rise_time_manual = t(t_90_idx) - t(t_10_idx);
    else
        rise_time_manual = 0;
    end
else
    rise_time_manual = info.RiseTime;
end

% Display summary
fprintf('--- ANFIS Performance Summary ---\n');
fprintf('Rise Time: %.5f s\n', rise_time_manual);
fprintf('Overshoot: %.2f %%\n', info.Overshoot);
fprintf('Steady-state value: %.5f\n', omega_anfis(end));
fprintf('Mean Squared Error (MSE): %.5f\n', mse);
```

5.0 PERFORMANCE EVALUATION & COMPARISON



Metric	**PID**	**Fuzzy Logic**	**ANFIS**
Rise Time (s)	0.1311	0.531	0.57
Settling Time (s)	0.2578	–	–
Overshoot (%)	1.004	0	0
Mean Squared Error (MSE)	0.00537	0.20822	0.35918
Steady-state Value	1	0.70567	0.49791

Based on the performance metrics, the PID controller outperforms both the Fuzzy Logic and ANFIS controllers for this DC motor system. The PID controller has the fastest rise time (0.13 s), lowest overshoot (1%), lowest mean squared error (0.00537), and perfectly reaches the setpoint (steady-state value of 1). This is expected because PID controllers are well-suited for linear systems like this DC motor model, providing quick and accurate responses.

The Fuzzy Logic controller also performs well, with no overshoot and a steady-state value of about 0.71, but it is slower to respond (rise time 0.53 s) and less accurate (higher MSE) than PID. This suggests it is stable but not tuned for perfect tracking. The ANFIS controller shows similar behavior to the Fuzzy controller, with no overshoot, but it is the slowest (rise time 0.57 s) and has the highest steady-state error (final value 0.50) and MSE (0.36). Indicates poor tracking and possibly undertrained or poorly tuned.

To compare, while intelligent controllers like Fuzzy Logic and ANFIS are powerful for handling nonlinear or uncertain systems, the classic PID controller is best for this linear DC motor, giving the fastest and most accurate results. However, in more complex or nonlinear situations, intelligent controllers may have advantages that are not seen in this simple test. Overall, the results make sense for a linear system which we found that PID outperforms, while Fuzzy and ANFIS need better tuning for steady-state accuracy.

6.0 CONCLUSION

In this project, we analyzed and compared the performance of three controllers which is PID, Fuzzy Logic, and ANFIS for controlling the speed of a DC motor. The PID controller demonstrated the fastest rise time (0.1311 seconds), minimal steady-state error, and the lowest mean squared error (0.00537), but exhibited a small overshoot (\~1%). This aligns with theory, as PID controllers are highly effective for linear systems but can produce overshoot due to their aggressive response.

The Fuzzy Logic controller eliminated overshoot entirely, showcasing its strength in handling nonlinearities and providing smoother control. However, it experienced steady-state error, settling at approximately 70% of the desired speed, resulting in higher MSE (0.20822) than PID. This outcome reflects the theoretical advantage of fuzzy logic in avoiding abrupt system responses but also highlights its need for careful rule tuning to achieve accurate tracking.

The ANFIS controller further avoided overshoot and offered adaptive capabilities by combining neural networks with fuzzy logic. Nevertheless, it performed the worst in terms of steady-state tracking, achieving only about half the target speed (0.49791) and recording the highest MSE (0.35918). This suggests that while ANFIS has potential for learning and adapting to nonlinear systems, it requires more extensive training and data to deliver precise control.

Theoretically, ANFIS should be even better than fuzzy logic since it can learn from data, but it underperformed here due to limited training and model tuning.To improve Fuzzy and ANFIS, we should collect more data, refine their designs, and train them better. Overall, PID was the best for this project because our DC motor problem was mainly linear.

7.0 CONTRIBUTION

NAME	**PART**
QASHRINA AISYA BINTI MOHD NAZARUDIN	DC Motor Modelling, ANFIS Training Simulation, Fuzzy logic Simulation
SHAREEFAH HUMAIRA BINTI BASHEERUDIN	Report Writing on ANFIS controller, Performance Evaluation, Conclusion
IZZAH ZAHIRA BINTI NORAZLEE	Report writing on PID Controller and Fuzzy logic Controller

ADIBAH BINTI MOHD AZILAN	Report writing on Problem Statement, Results Comparison

8.0 REFERENCES

1. Alfa Tron. (2021, October 31). *Simulasi logika fuzzy Sugeno dengan Matlab* [Video]. YouTube. https://www.youtube.com/watch?v=Rz-A3Os-nlE

2. EEprogrammer. (2012, May ? – approximate). *MATLAB tutorial – Fuzzy Logic* [Video]. YouTube. https://www.youtube.com/watch?v=-42QUDfdb9I

3. Åström, K. J., & Hägglund, T. (2006). *Advanced PID Control. ISA*–The Instrumentation, Systems, and Automation Society. https://www.isa.org/products/advanced-pid-control

4. Li, Y., Ang, K. H., & Chong, G. C. Y. (2006). PID control system analysis and design: Problems, remedies, and future directions. IEEE Control Systems Magazine, 26(1), 32–41. https://doi.org/10.1109/MCS.2006.1580159